View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

ABC-Fun: A Probabilistic Programming Language for Biology

Citation for published version:
Georgoulas, A, Hillston, J & Sanguinetti, G 2013, ABC-Fun: A Probabilistic Programming Language for Biology. in A Gupta & TA Henzinger (eds), Computational Methods in Systems Biology: 11th International Conference, CMSB 2013, Klosterneuburg, Austria, September 22-24, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8130, Springer-Verlag GmbH, pp. 150-163, The 11th Conference on Computational Methods in Systems Biology (CMSB 2013), Klosterneuburg, Austria, 23/09/13. DOI: 10.1007/978-3-642-40708-6_12

Digital Object Identifier (DOI):
10.1007/978-3-642-40708-6_12

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Computational Methods in Systems Biology

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Apr. 2019

Georgoulas, A., Hillston, J., & Sanguinetti, G. (2013). ABC-Fun: A Probabilistic Programming Language for Biology. In Gupta, A., & Henzinger, T. A. (Eds.), Computational Methods in Systems Biology. (pp. 150-163). (Lecture Notes in Computer Science). Springer Berlin / Heidelberg. doi: 10.1007/978-3-642-40708-6_12

ABC–Fun: A Probabilistic Programming Language for Biology

Anastasis Georgoulas[1], Jane Hillston[1,2], and Guido Sanguinetti[1,2]

[1] School of Informatics, University of Edinburgh
[2] SynthSys — Synthetic and Systems Biology, University of Edinburgh

Abstract. Formal methods have long been employed to capture the dynamics of biological systems in terms of Continuous Time Markov Chains. The formal approach enables the use of elegant analysis tools such as model checking, but usually relies on a complete specification of the model of interest and cannot easily accommodate uncertain data. In contrast, data-driven modelling, based on machine learning techniques, can fit models to available data but their reliance on low level mathematical descriptions of systems makes it difficult to readily transfer methods from one problem to the next. Probabilistic programming languages potentially offer a framework in which the strengths of these two approaches can be combined, yet their expressivity is limited at the moment.
We propose a high-level framework for specifying and performing inference on descriptions of models using a probabilistic programming language. We extend the expressivity of an existing probabilistic programming language, Infer.NET Fun, in order to enable inference and simulation of CTMCs. We demonstrate our method on simple test cases, including a more complex model of gene expression. Our results suggest that this is a promising approach with room for future development on the interface between formal methods and machine learning.

1 Introduction

Continuous Time Markov Chains (CTMCs) have long been established as a framework for the description and analysis of dynamical systems, including those encountered in the life sciences. Of particular interest, and also widespread, is their use within high-level formalisms, such as process algebras. Adopting a high-level language rather than working with the CTMC itself offers various advantages: a friendlier language to specify the system, easier modification and some degree of inbuilt error-checking, among others. It also gives access to an array of tools to analyse and reason about the behaviour of the system, such as stochastic simulation and model-checking. The major weakness of this framework is that the models implicitly assume complete mechanistic knowledge of the system. Therefore it does not offer support for integrating experimental data into models or inferring parameters from observations.

In contrast, machine learning techniques applied to models of biological processes are designed to predict a system's behaviour in the presence of uncertainty.

An important category of such methods is concerned with using (possibly noisy) observations from the system in order to refine our understanding of it, a task often referred to as *learning* and which can be broken down into two aspects: learning the structure of the system in question, or learning its parameters (for a given structure). These methods mostly work on mathematical descriptions of systems, often in the form of ordinary, partial or stochastic differential equations. For example, Bayesian Networks, a graphical framework for describing probabilistic models, while intuitive and widely used are still essentially a front-end to the underlying equations. Working with the low-level description negates the advantages afforded by high-level languages, as described above, thus limiting the applicability of the inference techniques.

Some common ground between the two approaches may be found in the field of *probabilistic programming*. Probabilistic programming promises to offer a high-level language that can be used for both describing and learning non-deterministic systems. Using a programming language and the expressive power it affords makes the the process of specifying a system easier, while at the same time offering a range of other features such as modularity and type systems. Additionally, we also obtain a unified, general framework for automatically performing inference on a given model, eliminating the need to write bespoke solutions and learning algorithms for every system of interest.

While the field of probabilistic programming has generated considerable interest in recent years, current probabilistic programming languages are limited in the types of models they can describe, as well as in the inference methodologies they implement. In particular, to our knowledge, continuous time, non-parametric models such as CTMCs cannot be handled by current probabilistic programs. Several techniques have been proposed, including approximations based on finite dimensional projections [1], sampling methodologies (e.g.[15, 3]), and variational inference approaches [12]. However, these methods all require a low level mathematical description of the system (usually a way of approximately solving the chemical master equation).

In this paper, we explore the potential for a framework which encompasses the strengths of both high-level formalisms and machine learning by extending an existing probabilistic programming language, Infer.NET Fun [2], in order to enable stochastic simulation and approximate Bayesian inference for CTMCs. We call the resulting approach ABC–Fun and illustrate it on two examples of biological significance, showing the potential of probabilistic programming as an effective tool for modelling in systems biology. Our focus, however, is not on presenting a fully-formed solution but rather on exploring the applicability of a novel approach, with the ultimate aim of facilitating interfacing of models and experiments.

The rest of the paper is structured as follows: we give an overview of probabilistic programming and the platform we are using, Infer.NET Fun; we briefly describe our implementation of Gillespie's stochastic simulation algorithm, the basis of our approach, before presenting the inference process; finally, we describe our experiments and discuss their results.

2 Background

2.1 Bayesian Inference

In this work, we focus on the *Bayesian* approach to learning, which uses probability distributions to model and quantify uncertainty about all aspects of the system under study, including its structure or parameters. Assume the system to be characterised by a set of parameter values Θ (e.g. transition rates of a certain CTMC). We are also given a set of (partial) observations of the system $\mathbf{y}$. The principal ingredients of the Bayesian approach are two: the *prior distribution* $p(\Theta)$ encodes any initial beliefs about the values of the parameters. The *likelihood* $p(\mathbf{y}|\Theta)$ (sometimes called observation model) gives the probability of the observations given the values of the parameters. Since the observations are fixed, this is a function of the parameter values. Bayes' rule combines these two ingredients to provide a mathematically sound way of estimating the impact of the observations on our beliefs over the parameters,

$$p\left(\Theta|\mathbf{y}\right) = \frac{1}{Z} p\left(\mathbf{y}|\Theta\right) p\left(\Theta\right). \tag{1}$$

$p\left(\Theta|\mathbf{y}\right)$ is the *posterior distribution* over the parameters, which quantifies the uncertainty over the parameters implied by the data and the prior beliefs.

A major computational hurdle in applying Bayes' rule is the estimation of the proportionality constant Z in equation (1). This term, the *marginal likelihood* or *evidence*, represents the probability of the data under all possible settings of the parameters; its value is obtained by performing (usually analytically intractable) integrals over the parameter space, which become prohibitive in even moderate dimensions. In the case of CTMCs, this problem is further compounded by the fact that (in general) even the likelihood cannot be computed analytically: the probability of the state of a CTMC taking a particular value at a certain time can only be obtained by solving the chemical master equation, which is impossible in most cases. In general, Bayesian inference in CTMCs remains a challenging problem: current methods either resort to approximations to the chemical master equations [12, 1] or sampling approximations [15, 3]. In all of these approaches, inference relies on a low level mathematical description of the system as a Markov transition system, and often specific characteristics of the system (e.g. functional form of the transition rates) are hard-wired in any accompanying code, greatly reducing the ease of portability and applicability of the approach.

2.2 Probabilistic Programming

Probabilistic programs can be thought of as an extension of conventional, deterministic programs, in which expressions describe stochastic experiments. Rather than having a concrete value, then, an expression corresponds to a whole distribution over values and evaluating it means performing the experiment and recording its outcome [13]. Constraining some variables within an expression to

have a specified value is equivalent to performing inference, with the observations representing constraints.

Historically, probabilistic programming languages have been primarily targeted at graphical models, a popular class of models in machine learning. Briefly, a graphical model is a specification of a finite number of random variables and the (conditional) dependence relationship which define their joint distribution. The name graphical model derives from the fact that such models can be represented as graphs or networks; this graphical representation enables a quick and intuitive formulation of the model, and also encodes several properties which are important for simplifying inference. For a thorough review, we refer the reader to the excellent book [7].

Examples of languages for probabilistic programming include IBAL [13] and Church [5]. Both of these use a functional language to specify probabilistic models, equipped with a way of performing inference on them. The generation of the inference code is automated and tailored to the model at hand, which means the user can focus on describing the model and not on adjusting or rewriting code for every different model. However, the automation of inference comes at a cost, either in terms of the class of models that can be considered (IBAL for example only considers finite graphical models), or of the inference methodology employed (Church only allows the Metropolis-Hastings sampling algorithm, which requires an analytically tractable likelihood function).

Infer.NET [9] is a probabilistic programming framework developed by Microsoft Research for specifying probabilistic models and performing Bayesian inference on them. More specifically, it offers a high-level, programming language interface for the description of graphical models. Further to this, Infer.NET includes an inference engine that can use a number of different algorithms, such as Expectation Propagation and Gibbs Sampling, to obtain estimates of the distribution of the model's parameters, informed by the knowledge of some observed data. Infer.NET also provides bindings for programming languages such as C# and Python and these can be used to describe a model and the desired inference queries, which are then compiled into source code. The resulting code can then itself be compiled and executed, returning the results of the inference queries. An additional component of Infer.NET is Fun ([2, 6]), an F# interface that aims to make the process of describing a model even more similar to "conventional" programming. As such, its syntax is very lightweight and consists of simple additions to the standard F# syntax, resulting in a user-friendly framework.

A model expressed in Fun can be used in two different yet related ways. The first is to view it as a generative model, that is, a description of how data points are generated. In this case, every `random` expression produces a sample from the given distribution. The model can therefore be "run forwards", giving rise to a set of values in a style similar to ancestral sampling.

The second way to use a model is to pass it to the Infer.NET inference engine. The model is then "run backwards". We use `observe` expressions to specify observed data and condition the model on them — when such an expression is encountered, if its condition is not met, the execution is marked as failed. The

result of this procedure is that we can obtain the posterior distributions on the random variables of the model, i.e. the distribution when only considering those executions which satisfy all the observations.

To illustrate what models probabilistic programming languages can handle, and see why they are not sufficient for CTMCs, we consider the kind of systems that can be modelled in Fun. In the simplest case, a Fun model can describe a static system with a finite number of random variables. In this case, one would describe how the variables depend on one another by specifying their conditional distributions. Some of these distributions may be parametrized, and the unknown parameters are also treated like random variables, in the Bayesian style.

A more complex example which can still be modelled in Fun is a dynamical system with a known number of steps. The state at each time depends only on the previous state, thus the system can be represented as a Markov Chain. Describing this would result in a recursive definition, which is not supported by Infer.NET. However, in the functional programming paradigm, this can be reformulated as a folding operation, to avoid explicit recursion. Folding a function over an array involves applying it to all elements of the array sequentially. The result of every application is used to obtain a new function, which is then applied to the next element, and so on. This technique has been applied to one of the examples in [6], but it should be noted that this is made possible because the number of steps in the system is known *a priori*.

3 Probabilistic programming for CTMCs

In this section we highlight the limitations of Fun and describe how these can be addressed in an economic way by exploiting Fun's parent language, F#.

Infer.NET, and Fun in particular, is designed to address graphical models, i.e. models with a known, finite number of random variables. In particular, the number of random variables is hard-wired in the definition of a Fun model type. In the case of CTMCs, however, the number of transitions that may occur in a given time is generally unknown, therefore so is the number of random variables (since every transition is associated with two random variables, one each for the choice of transition and delay). This means both that we do not have an array to fold over, and so cannot eliminate the recursion, and that we must use lists instead of arrays, since the latter have a fixed size while the former can grow indefinitely. However, these are not features supported by Fun or Infer.NET, so we must leave Fun for a different language. A major implication of this is that the inbuilt inference engine of Infer.NET cannot be used, so that alternative inference strategies need to be used (detailed in Section 3.2).

As we would like to retain some of the functionality of Fun, such as drawing samples, we turn to its base language, F#, and use Fun as a library for the operations we need. This way, we can retain useful language structures and features such as random number generators for various different probability distributions, although the final code will not be compatible with the Infer.NET inference engine. Moreover the F# code has been packaged as a library for FUN making

it available to other users who are not necessarily familiar with the implementation details for CTMCs. This illustrates our aim to lift the data generation and inference techniques into a high-level language, supporting their use by a wide-range of users as transparently as possible.

3.1 Implementing the Stochastic Simulation Algorithm

The ability to simulate CTMCs is central for both modelling and inference. We describe here the ABC-Fun implementation of Gillespie's Stochastic Simulation Algorithm [4]; this is given in considerable detail both because of its importance and in order to provide the reader with a concrete example of ABC-Fun syntax.

We handle models as chemical reactions: a model with N species and K reactions is comprised of a stoichiometry matrix and a list of kinetic laws. The stoichiometry matrix is implemented as a list of lists; each of its K sub-lists has length N, corresponds to a reaction and contains the updates for the population of each species when that reaction occurs. Each reaction also has an associated kinetic law, which is a function that acts on a list of integers (the state of the system) and returns a real value (the rate of the reaction); these are collected in the second component of the model. A model can also be parametrized – such a model is essentially a function that takes a list of parameters and returns a concrete model, as described above. We define a `Model` constructor, which combines the two elements (list of rates and stoichiometry matrix) and creates a `model` object. Technically, this has the F# type:

```
(int list -> float) list * (int list) list -> model
```

This provides an interface through which one can specify models, without concern for how the simulation is performed. For example, a model of a single species birth-process can be encoded as follows:

```
let r1_1 l = 0.1 * float (List.head l)
let r1_2 l = 20.0     // constant rate
let rateLaws1 = [r1_1; r1_2]
let stoich1 = [[-1]; [1]]
let m1 = Model(rateLaws1,stoich1)
```

A trace (one possible run of the CTMC) can then be obtained simply by calling the function `pathSample`, which accepts a model, an initial state and the stopping time, and returns a trajectory through the state space (a list of states and a list of the corresponding times). Its type is therefore

```
model -> int list -> float -> int list list * float list
```

In order to generate a trace, we must make explicit the sampling steps involved in the SSA. To do so in a probabilistic programming language, we use some of the Fun in-built functions (primarily the random number generators). The SSA can be recast in probabilistic programming terms if we consider that, at every step, the next reaction to occur is a discrete random variable, with each possible value having a probability that can be calculated from the reaction

rates at the current state. Similarly, the time to the next state is also a random variable. Both of these probabilities are encoded in the kinetic parameters of the model. In order to simulate the CTMC, we keep the parameter fixed; we will see in the next section how to vary the parameters in an inference scheme. The code below shows how we implement the standard version of the SSA in F# using Fun as a library. The part shown here specifies how the next reaction and delay are chosen. It is straightforward to use this in order to recursively build the trajectory, keeping a list of the states and transition times and stopping when we reach the final time.

```
let nextStateAndTime state rateFunctions stoich =
  let rates = [|for r in rateFunctions -> r state|]
  let sumRates = Seq.sum rates
    if sumRates > 0.0 then
      let delay = ExponentialSample sumRates
      let reaction =  random(Discrete (Vector.FromArray rates))
      let newState = updateState state (List.nth stoich reaction)

      (newState,delay)
    else //all rates are 0, we can stop
      (state, infinity)
```

Note that the code above uses the `random` construct from Fun to sample from a distribution. `ExponentialSample` is also defined using `random` to draw from an exponential distribution with the specified rate:

```
let ExponentialDist rate = GammaFromShapeAndRate(1.0, rate)
let ExponentialSample rate = random(ExponentialDist rate)
```

`updateState` simply calculates the next state given the current state and the row of the stoichiometry matrix corresponding to the chosen reaction. As both these arguments are represented as lists, this can be expressed as the pairwise sum of their elements.

3.2 Choosing an Inference engine

As explained earlier, the limitations of the Infer.NET engine mean that we must adopt a different inference method. For the purposes of this work, we use Approximate Bayesian Computation (ABC) [19], a parameter inference scheme that constructs an approximation of the posterior distribution by repeated simulations of the system. This enables us to use our implementation of the SSA to also perform inference, as described below.

ABC works by generating samples of parameters. For each such parameter, the behaviour of the system is simulated, in our case producing a path through the state space. If the path obtained this way matches the observed data, under some given metrics and tolerance, the parameter sample is kept, otherwise it is discarded. The process is repeated until a sufficient number of samples has been accepted, and the resulting set of accepted parameter values serves as the approximate representation of the posterior. ABC can therefore be thought

of as a way of converting simulation into an inference technique. The choice of tolerance can be significant, as there is a trade-off between accuracy and efficiency of the algorithm. The lower the tolerance, the harder it is to accept a sample, which means more sampling attempts will be required in order to reach the desired number of accepted samples (as the rejection rate will be higher) but the final set will be more representative of the true posterior distribution.

There are different versions of the algorithm (as described, for example, in [17]), depending on how the parameter samples are generated. The simplest approach is to sample independently from a prior distribution, which may however prove to be inefficient, while another is to have each sample depend on the previous one, giving rise to a Markov Chain Monte Carlo (MCMC) scheme.

Choosing a metric to evaluate the distance between a simulated trace and the observed data is an important issue. Assume we have a series of successive observations along with the corresponding measurement times $\{(y_1, t_1), ..., (y_N, t_N)\}$, and a simulated trace $\{(x_1, \tau_1), ..., (x_M, \tau_M)\}$, with $M > N$[3]. In this work, the first thing we do is "shift" and thin the trace, keeping the value at every t_i. Formally, we define a new time-series $\hat{x}$ such that $\hat{x}_i = x_m$, where $m = argmax_j(\tau_j \leq t_i)$, for $i = 1, .., N$.

The simplest way to calculate the distance between x and y is to take the absolute difference between $\hat{x}$ and y, averaged over all points:

$$d(x, y) = \frac{1}{N} \sum_{i=1}^{N} |\hat{x}_i - y_i| \tag{2}$$

In the case of CTMCs, a plausible alternative could be to rescale the difference between $\hat{x}$ and y adaptively according to the value of y; this can be justified by noticing that noise in CTMCs is usually multiplicative. We therefore also consider the following metric

$$\tilde{d}(x, y) = \frac{1}{N} \sum_{i=1}^{N} \frac{|\hat{x}_i - y_i|}{\sqrt{y_i}} \tag{3}$$

which may be more suitable when the observations span a wide range of values.

In this work, we perform parameter estimation using the MCMC version of the ABC algorithm. This works by constructing a Markov chain in parameter space which asymptotically will converge to an approximation of the posterior distribution. Given data and an initial set of parameter values, we sample a new set of parameter values from a proposal distribution which depends on the current parameters (in our application, a Gaussian centred at the old parameter values). We then simulate the system (by running the model forwards using these values), compare the trace obtained this way with the input data and decide whether to accept it or not, depending on the "distance" between the two traces, our tolerance level and the prior and proposal distributions. We

[3] Notice that our measurements are counts at individual times, not transition times; therefore, an analytical expression for the likelihood is in general not available.

repeat this sampling scheme for a previously specified number of steps, then use the samples collected this way as a representation of the posterior distribution of the parameters.

4 Experiments

4.1 Birth-death process

We have tested ABC–Fun on a simple model of a birth-death process, in which a single species can be created at a constant rate or degraded according to mass-action kinetics (i.e. at a rate proportional to its amount). We initially fix the death rate and reduce the problem to a one-dimensional one, where we try to estimate the birth rate. We use a uniform prior, reflecting a belief that all parameter values are equally likely in the absence of any data.After 10^5 steps, the distribution of samples is clearly centred around the true parameter value (Figure 1). A common issue with MCMC schemes is the difficulty of assessing whether the process has converged, i.e. whether the samples are truly representative of the posterior distribution and have "overcome" the influence of the initial state. Experimenting with different initial samples indicated that this convergence to the true value was robust, although it required more samples when the initial point was further away from the true value (Figure 2).

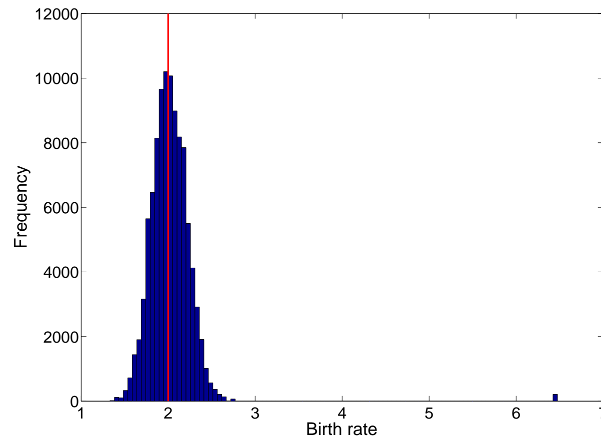


Fig. 1. Histogram showing the number of accepted samples after 100000 steps of the ABC-MCMC algorithm when inferring only the birth rate. The true value of the parameter is 2, shown by the vertical line.

We then considered the problem of inferring the full model, i.e. estimating both rates. The parameters chosen for the simulation were a birth rate of 2

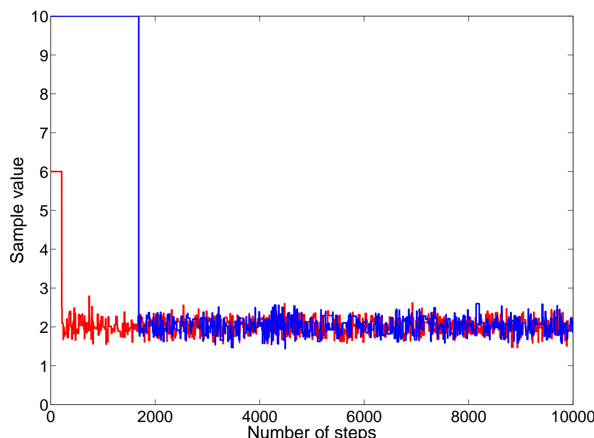
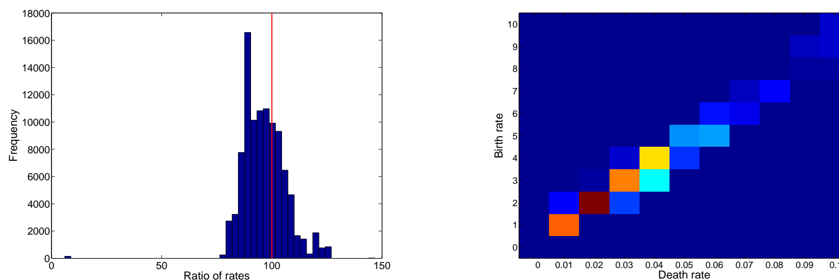


Fig. 2. Accepted samples when inferring only the birth rate, showing convergence to the true value. The algorithm was run for 10000 steps, starting from an initial value of 6 (red) or 10 (blue). Similar results occur for all tested initializations.

and a death rate of 0.02, leading to the steady state of 100 being reached after about 250 time units. The data that was used as input for the inference came from a single stochastic simulation, taking 20 samples from the resulting trace (approximately 15 during the transient phase and 5 at the steady state).

This time, the results are not as clear-cut: the heat map (Figure 3b) indicates there are multiple value pairs that match the observed data. This reflects an identifiability issue with the system, which can be easily explained if we consider that the probability of choosing one reaction over the other depends only on the ratio between the two rates. Therefore, there exist multiple parameterizations which would give the same relative probabilities, and the unknown parameters can only be estimated up to a multiplicative constant. It is possible to reduce this uncertainty by considering information about the timing of the reactions, as the duration of the reactions does depend on the concrete values of the parameters, rather than just their ratio — intuitively, higher rates will result in faster reactions.

The results show that the highest number of accepted samples is concentrated around the true values of the parameters. Additionally, the other areas with a significant number of samples lie on a diagonal line, indicating a constant ratio between the two parameters, matching our expectation. If we plot the ratio of the two parameters for each of the accepted samples (Figure 3a), we can see that this quantity is most often close to 100, its true value. In short, the inference procedure manages to distinguish the true parameter pair from the others that give a similar behaviour.



(a) histogram of birth to death rate ratio for the accepted samples (true value: 100)

(b) heat map of accepted samples (true values: 2 and 0.02)

Fig. 3. Accepted samples when inferring both kinetic rates in the new experimental configuration

4.2 Regulation of gene expression in single cells

As a more biologically meaningful example, we consider the classic on/ off model of gene expression (e.g. [14]). Here, the rate of mRNA production is assumed to depend solely on the state of the gene promoter: thus, mRNA can be produced at a high rate (promoter ON) or a low rate (promoter OFF). The inference task is to reconstruct both mRNA production/ decay rates and the promoter occupancy state from gene expression time series. This model was recently used in [18] to tease apart bursting kinetics in mRNA production; there the parameters were estimated by maximum likelihood. Bayesian inference methodologies for this model have been recently proposed assuming mRNA concentrations to be continuous variables [16, 11]; here, we consider the Bayesian inference problem when mRNA counts are discrete, and are thus governed by a birth/ death process whose birth rate depends on an unobserved binary process[4]. The importance of this model lies not only in its fundamental role as a mechanism for gene expression, but also in the possibility of using it as a building block for modelling complex gene regulatory networks [10].

To slightly simplify the task, we assumed that the promoter state only performed two transitions within the time frame under consideration (i.e., it starts in the OFF condition, turns ON at a random time, and then turns OFF again). We then tested the ABC–Fun approach on simulated data under ten different configurations of the model parameters/ switching times. For these experiments we used the modified distance metric $\tilde{d}$ (Equation 3). Figure 4 shows the posterior probabilities of the ON and OFF times in a particular run, with the true values indicated by a vertical line. As we can see, the posterior distribution is approximately centred around the true value. The inferred posterior distributions for the promoter activity (difference between birth rate in the two states)

[4] This can be seen as a special case of Bayesian inference for Markov Jump Processes [12, 20], albeit employing a different inference methodology.

and the decay rate are shown in Figure 5. We can see that the posterior distribution has substantial mass concentrated around the true value, but is quite wide due to the identifiability problems already mentioned in Section 4.1. Results for other configurations of the parameters gave qualitatively similar results.

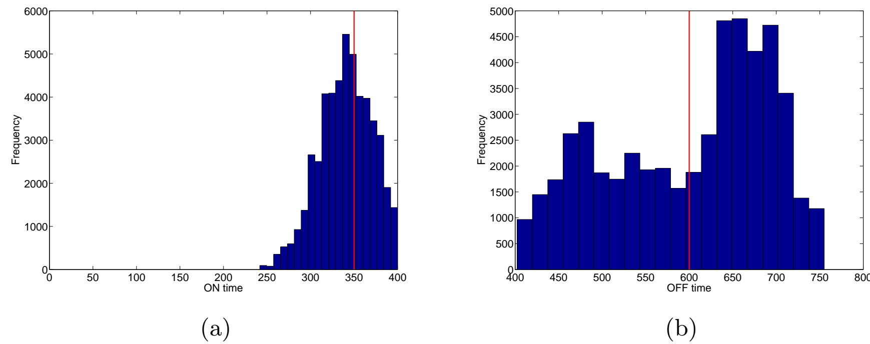


Fig. 4. Accepted samples when inferring the (a) switch-on and (b) switch-off time (real value shown by red vertical line)

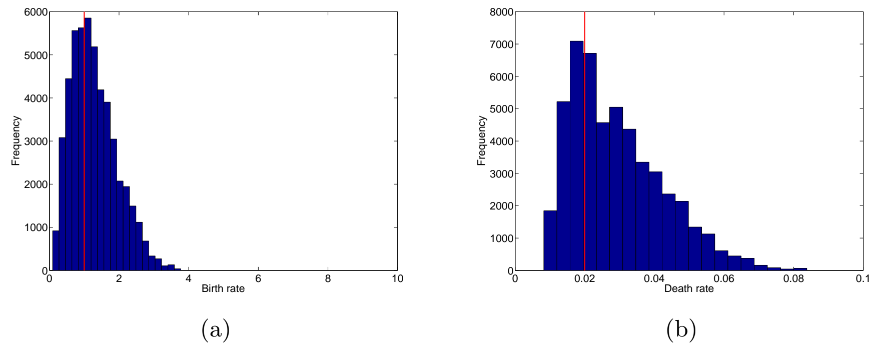


Fig. 5. Accepted samples when inferring the (a) promoter activity A) and (b) degradation rate λ (real value shown by red vertical line)

5 Discussion and conclusions

We have presented ABC–Fun, a probabilistic programming language which handles biological models expressed as CTMCs with uncertain rates. Our approach

uses features of an existing probabilistic programming language, Infer.NET-Fun; however, the non-parametric nature of CTMCs cannot directly be handled by Fun, so that we have to extend it by defining new types in F#, and use a different inference engine to perform approximate Bayesian inference. Our initial results on two simple but biologically relevant models show that this approach can be a valuable addition to the systems biology toolkit: in particular, the two different models only required minimal coding changes. We expect this high portability to be an increasingly important feature as systems biology matures to handle ever more complex models.

Our method extends the range of systems and inference methodologies that can be modelled using probabilistic programming languages, in addition to providing a test case for applying the latter in a biological context. We note that semi-automated inference packages using ABC have been proposed before: for example, [8] will take as input an SBML file and perform ABC-based inference on model parameters. Nevertheless, their approach is not based on a probabilistic programming language, and this has drawbacks: for example, it is not easy to express in SBML models with latent variables like the ON-OFF model of gene expression. In a probabilistic programming environment, this is straightforward as it is merely the addition of a further random variable.

Our choice of ABC as an inference engine was primarily motivated by its implementation ease and its applicability in intractable likelihood problems (such as CTMCs). Nevertheless, ABC has several drawbacks, both in terms of computational efficiency, and in terms of relying on a tolerance parameter which is difficult to tune in a principled way. Exploring alternative inference approaches which can ameliorate these problems will be key to extending our methodology to larger, more relevant models.

Acknowledgements This work was supported by Microsoft Research through its PhD Scholarship Programme. JH acknowledges support from the EU FET-Proactive programme through QUANTICOL grant 600708. GS acknowledges support from the European Research Council through grant MLCS306999. The authors would like to thank Luca Cardelli and Andy Gordon for useful discussion.

References

1. Andreychenko, A., Mikeev, L., Spieler, D., Wolf, V.: Approximate maximum likelihood estimation for stochastic chemical kinetics. EURASIP Journal on Bioinformatics and Systems Biology **2012**(1) (2012) 9
2. Borgström, J., Gordon, A.D., Greenberg, M., Margetson, J., Van Gael, J.: Measure transformer semantics for Bayesian machine learning. In: Proceedings of the 20th European conference on Programming languages and systems: part of the joint European conferences on theory and practice of software. ESOP'11/ETAPS'11, Berlin, Heidelberg, Springer-Verlag (2011) 77–96
3. Boys, R., Wilkinson, D., Kirkwood, T.: Bayesian inference for a discretely observed stochastic kinetic model. Statistics and Computing **18** (2008) 125–135

4. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry **81**(25) (1977) 2340–2361
5. Goodman, N.D., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: a language for generative models. In: UAI. (2008) 220–229
6. Gordon, A., Aizatulin, M., Borgström, J., Claret, G., Graepel, T., Nori, A., Rajamani, S., Russo, C.: A model-learner pattern for Bayesian reasoning. In: Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages (POPL). (2013)
7. Koller, D., Friedman, N.: Probabilistic Graphical Models. MIT Press, Cambridge MA (2010)
8. Liepe, J., Barnes, C., Cule, E., Erguler, K., Kirk, P., Toni, T., Stumpf, M.P.: ABC-SysBio—approximate Bayesian computation in Python with GPU support. Bioinformatics **26**(14) (2010) 1797–1799
9. Minka, T., Winn, J., Guiver, J., Knowles, D.: Infer.NET 2.5 (2012) Microsoft Research Cambridge. http://research.microsoft.com/infernet.
10. Ocone, A., Millar, A.J., Sanguinetti, G.: Hybrid Regulatory Models: a statistically tractable approach to model regulatory network dynamics. Bioinformatics **29**(7) (2013) 910–916
11. Opper, M., Ruttor, A., Sanguinetti, G.: Approximate inference for Gaussian-jump processes. In: Advances in Neural Information Processing Systems 24. (2010)
12. Opper, M., Sanguinetti, G.: Variational inference for Markov jump processes. In Platt, J., Koller, D., Singer, Y., Roweis, S., eds.: Advances in Neural Information Processing Systems 20. MIT Press, Cambridge, MA (2008) 1105–1112
13. Pfeffer, A.: The Design and Implementation of IBAL: A General-Purpose Probabilistic Language. In Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. The MIT Press (2007)
14. Ptashne, M., Gann, A.: Genes and signals. Cold Harbor Spring Laboratory Press, New York (2002)
15. Rao, V., Teh, Y.W.: Fast MCMC sampling for Markov jump processes and continuous time Bayesian networks. In: UAI. (2011)
16. Sanguinetti, G., Ruttor, A., Opper, M., Archambeau, C.: Switching regulatory models of cellular stress response. Bioinformatics **25**(10) (2009) 1280–1286
17. Sisson, S.A., Fan, Y., Tanaka, M.M.: Sequential Monte Carlo without likelihoods. Proceedings of the National Academy of Sciences **104**(6) (2007) 1760–1765
18. Suter, D.M., Molina, N., Gatfield, D., Schneider, K., Schibler, U., Naef, F.: Mammalian genes are transcribed with widely different bursting kinetics. Science **332**(6028) (2011 Apr 22) 472–474
19. Toni, T., Welch, D., Strelkowa, N., Ipsen, A., Stumpf, M.P.: Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. Journal of The Royal Society Interface **6**(31) (2009) 187–202
20. Zechner, C., Pelet, S., Peter, M., Koeppl, H.: Recursive Bayesian estimation of stochastic rate constants from heterogeneous cell populations. In: CDC-ECE, IEEE (2011) 5837–5843